Fast Simulation of Interacting Carriers in Nanosimulators

Pieter Stroobant
IDLab, Ghent University - imec
Ghent, Belgium
pieter.stroobant@ugent.be

Luca Felicetti
Department of Engineering,
University of Perugia
Perugia, Italy
ing.luca.felicetti@gmail.com

Wouter Tavernier
IDLab, Ghent University - imec
Ghent, Belgium
wouter.tavernier@ugent.be

Didier Colle
IDLab, Ghent University - imec
Ghent, Belgium
didier.colle@ugent.be

Mauro Femminella
CNIT, UdR Perugia
Department of Engineering,
University of Perugia
Perugia, Italy
mauro.femminella@unipg.it

Gianluca Reali
CNIT, UdR Perugia
Department of Engineering,
University of Perugia
Perugia, Italy
gianluca.reali@unipg.it

Mario Pickavet
IDLab, Ghent University - imec
Ghent, Belgium
mario.pickavet@ugent.be

ABSTRACT

Diffusion-based molecular communication (DMC) with interacting carrier molecules allow to explore a variety of new communication paradigms. However, simulating these interactions on a network scale is hard from an analytic point of view, and a computationally challenging task which may easily become a bottleneck. This paper studies different techniques that allow to detect which particles are interacting. Recursive and hierarchical grid based approaches are proposed and a multithreaded CPU and GPU implementation respectively are evaluated and compared to a state-of-the-art collision detection library and nanosimulator.

CCS CONCEPTS

• **Networks** → **Network simulations**; • **Theory of computation** → **Massively parallel algorithms**; *Computational geometry*; • **Applied computing** → *Biological networks*;

KEYWORDS

collision detection, multi-level grids, nanocommunication

ACM Reference format:
Pieter Stroobant, Luca Felicetti, Wouter Tavernier, Didier Colle, Mauro Femminella, Gianluca Reali, and Mario Pickavet. 2018. Fast Simulation of Interacting Carriers in Nanosimulators. In *Proceedings of NANOCOM '18: ACM The Fifth Annual International Conference on Nanoscale Computing and Communication, Reykjavik, Iceland, September 5–7, 2018 (NANOCOM '18),* 6 pages.

NANOCOM '18, September 5–7, 2018, Reykjavik, Iceland

ACM ISBN 978-1-4503-5711-1/18/09…$15.00
https://doi.org/10.1145/3233188.3233200

https://doi.org/10.1145/3233188.3233200

1 INTRODUCTION

Recent biology and medicine advances have sparked a growing interest in intra-body micro- and nanoscale communications [12]. In diffusive communication, transmitters encode information by releasing chemical signals. Various signals are transferred by modulating the concentration, timing or types of released molecules [21].

Many studies focus on transmission models in which the carriers are inert (i.e. not interacting in any way that affects the communication), but recently, transmission schemes in which different, interacting carrier molecules are released have been proposed [10, 11, 17]. This paradigm allows to explore a variety of new ideas: new waves of carriers signals may be created through in-channel reactions, and transmitted signals may interact with each other. This last idea is explored in [10, 17], where information is modulated by releasing bursts of acids and bases, which results in changing pH levels at a receiver. This method has been shown, both theoretically [10, 17] and experimentally [11] to significantly reduce intersymbol interference. Additionally, it has the benefit of reducing the contamination of the environment, which occurs when inert carriers are released.

There are other scenarios in which the impact of interactions between emitted particles cannot be ignored. For example, when the density of the transmitted molecules is high, or when the diffused particles attract or repel each other, the collective diffusion behaviour may be affected [22].

The main challenge in the evaluation of transmission schemes with interacting particles is caused by the non-linear communication channel. The underlying reason for this is that the diffusion equations that describe the system are coupled and non-linear. Calculating solutions for them is computationally expensive, even in 1D scenarios with a single transmitter [10]. On the other hand,

nanonetwork scenarios may feature thousands of communicating nanomachines [4]. This combination of factors complicates the calculation of the received signals.

Particle-based nanosimulators, such as BiNS2 [13, 14] and N3Sim [23], take the effect of these interactions into account by tracking the positions of all particles that may interact. Both the transceivers and released particles are enclosed by a bounding volume, and interactions between pairs of objects cay occur only if the corresponding bounding volumes are overlapping. At each simulated timestep, the simulator checks which particles are interacting/enclosing spheres are colliding, and resolves these interactions. This search for colliding bounding volumes is called broad-phase collision detection, and can easily become a bottleneck for the simulator.

Starting from a definition of a typical nanonetwork setting and an overview of related work, this paper proposes several algorithms for finding collisions between these bounding volumes. Proposed nanonetworking applications feature a large number of transceivers, possibly resulting in millions of nano-objects (i.e. both transceivers and released particles) [4]. With this in mind, we pay attention to parallelism and aim to significantly improve upon the collision detection times of existing nanosimulators. The performance of the resulting algorithms is measured and compared to that of Bullet, a widely used real-time collision detection library [1], and to the collision detection algorithm that is implemented in the BiNS2 simulator.

2 RELATED WORK

Whilst there is a large body of literature on collision detection [9], molecular communication is characterized by several properties that are distinct from typical collision detection scenarios:

- the majority of the objects are moving: datastructures must either allow fast updates, or efficient construction (in that case they can be rebuilt for each timestep);
- the volume in which objects interact with each other allows for a tight fit of a bounding sphere: chemical interactions occur when objects are within a certain binding radius of each other [5], furthermore the elongation of nano-objects and biological cells is usually limited and bounding spheres allow for a reasonably tight fit;
- object types can be of widely varying sizes: for example, transceivers can exceed the size of the emitted particles by several orders of magnitude;
- the number of objects greatly outnumbers the number of different object types and size variation within a single object type is limited: a simulation may contain several types of nanomachines/cells, but machines of the same type are similar in size;
- no assumptions can be made on the distribution of objects: they may be spread out evenly throughout the environment, but they might also be concentrated in certain areas

We also take a look at the existing work on nanoscale diffusion simulators, and how they handle interparticle collision detection:

- **BiNS2** - Biological and Nano-Scale Communication Simulator v.2 is a multi-threaded simulator of MolCom systems developed at the University of Perugia [13]. Its design includes a set of customization tools for creating objects and allows to model the behaviour and interactions of biological entities. The simulator employs a sort-and-sweep collision detection algorithm and octrees [15] to detect both interparticle collisions and collisions between particles and transceivers. Additionally, it allows to simulate diffusion-based and flow-based propagation models, into both constrained and open space environments. A more detailed description of the octree algorithm of this Java-based simulator is provided in subsection 4.3.
- **N3Sim** is a Java-based simulator of diffusion-based molecular communications. It allows analyzing molecular networks with several transmitters and receivers [23]. Elastic collisions between carriers can be simulated, and are detected using a sort-and-sweep method (Baraff's algorithm).
- **Smoldyn** is a program that allows to simulate cell scale biochemical simulations [2]. Although Smoldyn was not developed with the specific purpose of molecular communications in mind, bimolecular reactions may occur when molecules are within each other's binding radius and molecules can be added and removed on-the-fly, thus allowing to simulate interacting networks. Smoldyn employs uniform grids to partition the simulated environment and detect which objects may interact. This approach has been successfully accelerated using GPUs [8].
- **Simulators based on NS-2 and NS-3**: NS-2 and NS-3 are discrete-event network simulators, which were not originally designed for modelling MolCom systems. These simulators are organized in different software libraries that can work together. Their flexible structure has allowed implementing some basic MolCom elements. User programs can be written in either C++ or Python programming languages. Among others, NanoNS [16] is an NS-2 based simulator for diffusive molecular communication in aqueous media. Rather than detecting interactions between individual particles, it implements the multi-particle lattice gas automata algorithm, which divides the propagation medium into lattice sides/voxels. NanoNS does not support interactions between carrier objects. NS-3 based simulators such as nanoNS3 [18] rely on channel models, rather than particle based simulation. However, as discussed in section 1, modelling interactions between carriers results in a non-linear channel, which complicates the calculation of channel responses.
- **NCSim - Bacteria Nanonetworks** is a comprehensive simulation framework for molecular communications utilizing flagellated bacteria as carriers for information delivery [6]. These bacteria are able to interact and pass information through the process of conjugation. A (uniform) grid is employed to detect whether conjugation is possible. The computationally expensive simulation modules are implemented in C++, but generation of scenarios is possible through Python.
- **HLA Simulator**. In [3], the authors introduce the principles of design of a MolCom simulator that focuses on scalability, by adopting the high level architecture (HLA) model. This model is used to design a distributed simulation tool for MolCom, so that different scalability options can be used to add processing power and reduce the execution time. Collision

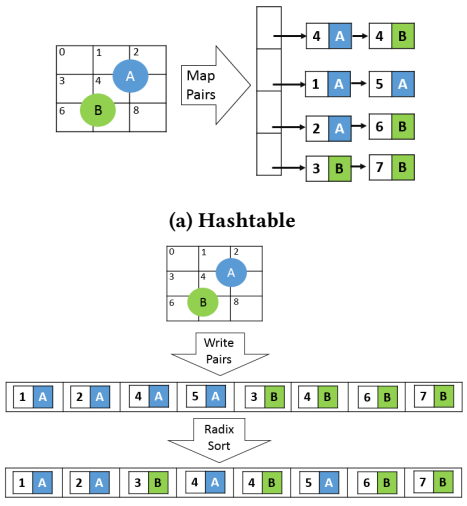
handling is managed by 'medium federates', which are responsible for collision detection and handling within a slice of the medium, which is divided according to a uniform grid.

3 ALGORITHMS

3.1 Uniform grids

Based on the requirements concerning update/build costs and the object shapes, partitioning the environment with uniform grids is a straightforward approach. The simulated space is overlaid with a grid and each object is mapped to all grid cells in which the object is partially present. Colliding object pairs can be found by iterating over all grid cells and verifying whether there is a collision between objects that are mapped to the same cell. Grids have very low construction times and combine well with spherical object shapes [9].

Since the objects may be spread out over a large environment, we cannot use a dense array to store the grid cells, as such an approach might may waste large amounts of memory to store empty cells. Therefore, the 3D rastercell coordinates are hashed, and information is maintained only for grid cells that do contain some objects. We implement two datastructures that achieve this goal. In our '*HashGrid*' approach (Fig. 1a), a hashtable with separate linked list chaining [7] stores objects that are mapped to the same cell, or whose hash is mapped to the same entry of the table. Another approach fills an array with (*cellId*, *objectId*)-pairs, corresponding to the grid cells that overlap with each object (*ArrayGrid*). By sorting this array on the *cellId*, the objects with the same cellId are stored after each other, and any *cellId* can quickly be retrieved by employing a binary search. Figure 1b illustrates this approach. This technique is well-known in the context of grids for ray tracing on GPUs [19, 20], but we are unaware of any multi-core CPU implementations.



(a) Hashtable

(b) Sorted array

Figure 1: Datastructures for storing object cell information

Both algorithms can easily be parallelized: the size of the hashtable can be precalculated and objects that are mapped to the same slot can be efficiently chained by employing lock-free linked lists. The array representation can be efficiently sorted using a multi-core radix sort [24].

3.2 Multi-level grids

The thus far discussed uniform grids have a single rastersize, which makes them inefficient for objects of widely varying sizes. A rastersize which is large compared to an object size may result in many small objects being mapped to the same rastercell, resulting in many collision checks. On the other hand, a grid size that is too small will cause each object to be mapped to a large number of cells, increasing the raster construction time.

Recursive grids solve this problem by mapping all objects to a grid with a size that corresponds to the largest object size. This grid is used for finding collisions that involve at least one large object. If the number of smaller objects that are mapped to the same rastercell exceeds a threshold τ, the cell is recursively split into a finer-grained grid. If the number of objects that are mapped to the same cell is smaller than, or equal to this threshold, collision checks are performed between all pairs. An example recursive grid is shown in Fig. 2a.

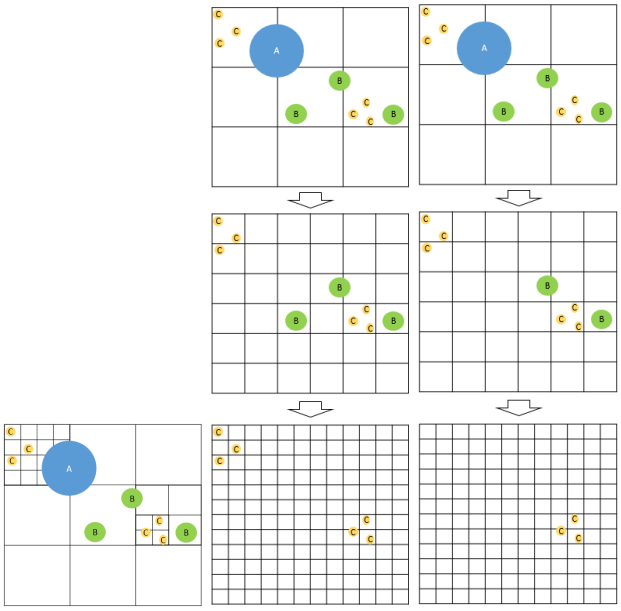
In a recursive grid, the depth of the recursive splitting, and the amount of work performed per cell may differ, depending on the object distribution and the specific number of objects per cell. In a multithreaded approach, where each thread processes one rastercell this will cause the execution paths of each thread to diverge. This is not a problem for CPU multithreading, but it is highly undesirable for GPU performance.

Therefore, our GPU approach uses a *hierachical* grid approach: rather than creating a smaller grained grid for each rastercell separately, all the smaller sized objects are bundled when constructing the smaller grid (EagerGrid). We also implemented an opportunistic/lazy version of this algorithm (LazyGrid). In this variant, smaller sized objects that are mapped to the same rastercell are not passed to the smaller level grid, if their number is smaller than or equal to a threshold τ. In that case, a brute force approach is used to detect the collisions between the smaller sized objects. If small sized objects are equally distributed, this technique may avoid the construction of smaller grained grid levels altogether. Both hierarchical grid approaches are illustrated in Fig. 2.

Complexity In these grid methods, the rastersize is chosen proportionally to the diameter of the largest object. This ensures that each bounding sphere is mapped to a limited number of grid cells. Thus, by applying any of the linear grid construction methods proposed before, we can construct a single grid level in linear time.

Additionally, assuming limited overlap, and assuming that the bounding spheres tightly fit their enclosed objects, the number of objects of the largest size are mapped to the same rastercell is limited by some constant c. Thus, for each rastercell that any object is mapped to, it must be compared with at most c objects of the largest size.

We conclude that both constructing a single grid level, and checking for collisions with large sized objects can be achieved in time $O(n)$, where n is the number of objects involved in the simulation. By ensuring that the gridsize at least halves for each finer grained



(a) Recursive grid ($\tau = 2$) (b) Eager hierarchical grid (c) Lazy hierarchical grid ($\tau = 2$)

Figure 2: Multi-level grid collision detection

grid level, we find a running time of:

$$O\left(\left(\log\left(\frac{\text{largest object size}}{\text{smallest object size}}\right)+1\right)n\right)$$

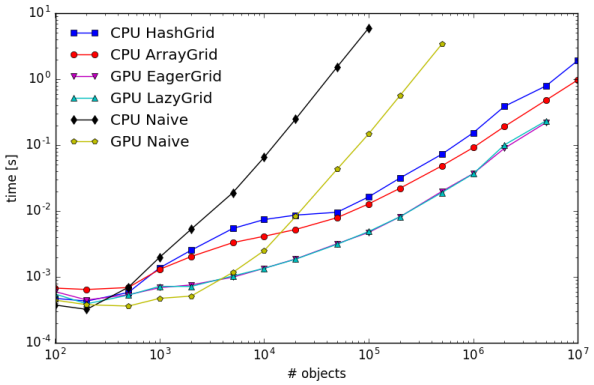
4 RESULTS

4.1 Comparison of proposed algorithms

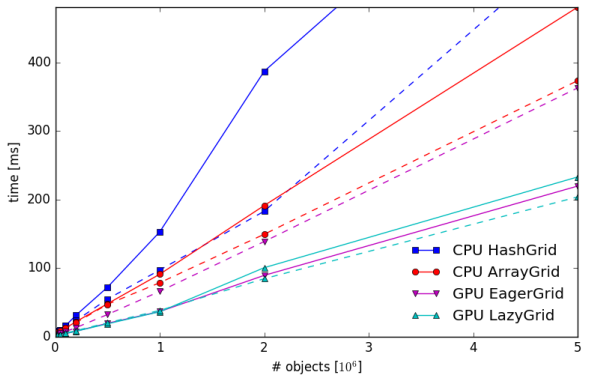
To evaluate the performance of the proposed algorithms for varying numbers of same-sized objects, we randomly generate scenarios with the required number of spherical objects, uniformly distributed in a spherical environment. The radius of the sphere is selected to maintain a constant density of 5% (i.e. a fraction of 5% of the volume is filled with nano-objects). Each datapoint visualises the average collision detection time of 200 runs. The resulting measurements, for an Intel Xeon E5645 processor, connected to an Nvidia Geforce GTX 580 are reported in Fig. 3a. The collision times for performing *naive* all-pairs collision detection are provided for comparison purposes.

The full lines in Fig. 3b visualise the same results and illustrate a linearly increasing detection time for the proposed algorithms. Furthermore, the arraygrid algorithm clearly outperforms the hashgrid algorithm. This is due to the memory access pattern of the hashgrid method: for a large number of objects, the hashtable cannot fit in cache memory and mapping objects to grid cells requires random accesses to RAM memory. For equisized objects, both GPU algorithms construct a single grid level, and thus achieve the same performance.

We also measured the collision detection time for scenarios with 5, equally frequent, object types of radius $1/2^0, 1/2^1, \ldots, 1/2^4$. Figure 3b compares the average computing time for 5 object types to the



(a) Equisized objects



(b) Equisized objects (full line) vs. 5 object sizes (dashed line)

Figure 3: Average collision detection time for varying numbers of objects

measurements with one object type. We notice that the collision detection time decreases for the recursive grid algorithms and the GPU lazy grid algorithm. This is because smaller sized objects are mapped to fewer raster cells, speeding up the raster building process. Additionally, these algorithms are able to avoid the construction of small grained grids. This is in contrast to the eager grid algorithm, which always needs to build as many grid levels as there are object sizes.

4.2 Comparison with Bullet Physics

For benchmarking purposes, we compare the performance of our collision detection algorithms to the Bullet Physics SDK [1], a well-maintained open-source C++ library for collision detection. Bullet's broad-phase collision detection algorithm uses a dynamic axis aligned bounding box (DAABB) tree. This collision detection datastructure is more complicated to create than the proposed multi-level grids, but it allows to exploit the spatiotemporal coherence of objects, resulting in very fast update times [9].

Bullet's DAABB tree is optimized for single thread collision detection. For a fair comparison, we compare the collision detection times with the performance of our own algorithm when executed

by a single thread. In each of the scenarios, spherical objects are uniformly distributed according to the procedure described in the previous subsection. For each test run, we generate a series of 100 updates in which the objects move by 10% of their radius in a random direction. These limited movements allow the DAABB tree to exploit spatiotemporal coherence. The resulting average collision detection times, for varying number of objects, are visualised in Fig. 4.

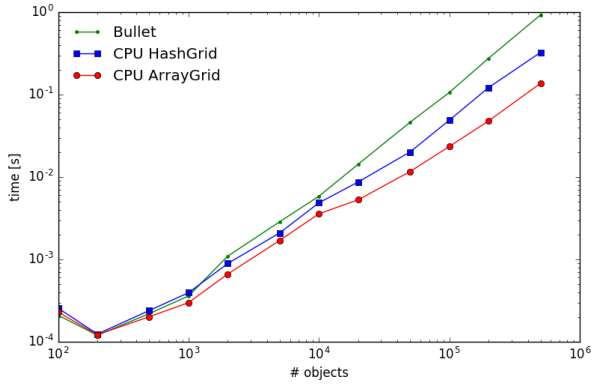


Figure 4: Average collision detection time for varying numbers of objects (single threaded)

Even in these test circumstances that allow the DABB tree to exploit spatio-temporal coherence, it is still outperformed by the recursive grids. In our test scenario, as in typical molecular communication scenarios, (almost) all objects are moving at all times, which makes updating the DABB tree more expensive than reconstructing a multi-level grid from scratch.

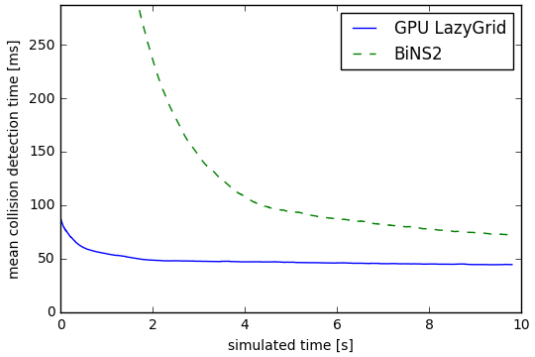
4.3 Comparison with BiNS2

In order to ensure that the proposed grids perform well in typical nanocommunication scenarios, the performance of the GPU accelerated LazyGrid detection algorithm is compared to the parallel octree algorithm that is implemented in BiNS2 [15]. Similarly to the DAABB tree, octrees aim to exploit spatiotemporal coherence by maintaining and updating object positions in a tree-based datastructure.

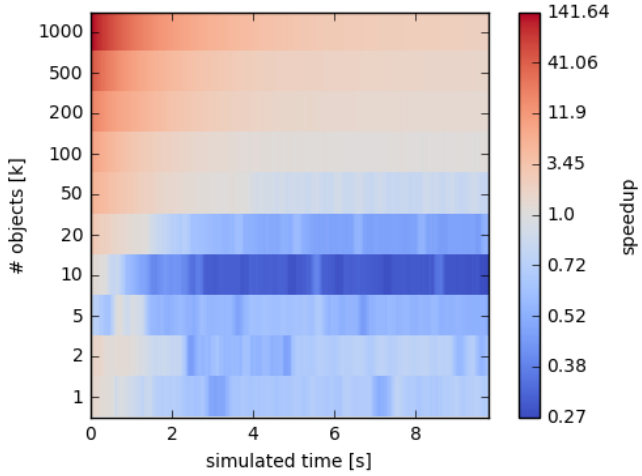
More specifically, the simulated environment is recursively partitioned in subdomains in order to model a hierarchical environment where some main domains can embed other subdomains. Each subdomain, and the objects therein, are handled separately from other subdomains. Within each of the 'leaf' subdomains, a parallelized sort-and-sweep algorithm is used. A collision detection time of $O(n \log(n))$ has been reported for this algorithm [15].

For comparison purposes, we consider a basic nanonetworking scenario: a single transmitter releases a single burst of interacting molecules in an unbounded environment. During the first seconds of simulated time, objects are close to each other and collisions are frequent. As time progresses, the number of carrier interactions quickly decreases.

In a first simulation, the average collision detection time for both the octree and the LazyGrid algorithm is measured, for a single burst of 5 10^5 carrier molecules of 10 different sizes. The results are visualised in Fig. 5a. For both algorithms, the collision detection time quickly decreases. In the case of the LazyGrid algorithm, this is due to the fact that all carriers are released at the same location. This violates the complexity requirement of limited overlap between carriers molecules. This issue is however quickly resolved by the inter-particle collisions. On the other hand, the octree algorithm requires a fine space partitioning when objects are clustered, resulting in frequent movements of objects from one part of the tree to another, and high update costs. As the simulation progresses, the density of objects decreases, allowing for a more coarse partitioning and less frequent updates.



(a) Average collision detection time vs. simulated time for a single burst of 500k objects of 10 different sizes



(b) Speedup of GPU ArrayGrid + GPU Naive vs. BiNS2 for a single burst of a varying number of objects (10 sizes) at varying moments in the simulation

Figure 5: Performance comparison with BiNS2

Figure 5b shows the speedup, i.e. the mean collision detection time of the octrees divided by the mean collision detection time of the proposed GPU LazyGrid approach, for a single burst of objects at varying times during the simulation. When fewer than 10^4 carrier molecules are released, the GPU naive algorithm is employed, since

it outperforms the GPU Lazygrid apploach in these cases (as shown in Fig. 3a).

The octrees are very efficient when the amount of interparticle interactions is small, i.e. as time progresses, and for simulations with a small number of objects. In these cases the octrees can be faster than the grid based approaches. However, when simulating large communication networks, or scenarios in which there are many interparticle interactions, the GPU LazyGrid outperforms the octree, sometimes by more than a factor of 100.

5 CONCLUSION

We proposed, implemented and evaluated several varieties of multi-level grid collision detection algorithms for nanocommmunication simulations, both on CPU and on GPU. In large networking scenarios, or in situations with many interparticle interactions, this approach outperforms other solutions by a significant margin, and has potential to accelerate the nanosimulators, allowing for more complex simulations and faster results.

6 FUTURE WORK

The hierarchical grids that are proposed in our work are in essence iterative constructions of increasingly finer grained uniform grids (with a decreasing number of objects). The construction of these uniform grids can effectively be parallelized using the ArrayGrid approach that is introduced in subsection 3.1. This work has explored the possibility of using a GPU, but other parallelization approaches, such as map-reduce, may also be interesting to explore.

We have focused on accelerating the collision detection process with a single GPU. For simulations with a large number of particles, expanding our approach to multi-GPU architectures or GPU clusters may cause significant performance improvements. Fully exploiting such systems without making any assumptions on the particle distribution is a challenging task.

ACKNOWLEDGMENTS

Pieter Stroobant is funded by a Ph.D. grant of Ghent University, Special Research Fund (BOF).

REFERENCES

[1] [n. d.]. Bullet Physics SDK: real-time collision detection and multi-physics simulation. https://github.com/bulletphysics/bullet3. ([n. d.]). Accessed: 2018-01-21.

[2] [n. d.]. Smoldyn: a spatial stochastic simulator for chemical reaction networks. http://www.smoldyn.org/. ([n. d.]). Accessed: 2018-01-21.

[3] Ali Akkaya, Gaye Genc, and Tuna Tugcu. 2014. HLA based architecture for molecular communication simulation. *Simulation Modelling Practice and Theory* 42, 0 (2014), 163 – 177.

[4] Ian F. Akyildiz, Fernando Brunetti, and Cristina Blázquez. 2008. Nanonetworks: A new communication paradigm. *Computer Networks* 52, 12 (2008), 2260 – 2279. https://doi.org/10.1016/j.comnet.2008.04.001

[5] Steven S Andrews and Dennis Bray. 2004. Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Physical Biology* 1, 3 (2004), 137. http://stacks.iop.org/1478-3975/1/i=3/a=001

[6] S. Balasubramaniam et al. 2013. Multi-Hop Conjugation Based Bacteria Nanonetworks. *IEEE Transactions on NanoBioscience* 12, 1 (March 2013), 47–59. https://doi.org/10.1109/TNB.2013.2239657

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms* (3 ed.). MIT Press and McGraw-Hill, Chapter 11, 221–252.

[8] Lorenzo Dematté. 2011. Smoldyn on Graphics Processing Units: Massively Parallel Brownian Dynamics Simulations. 9 (07 2011), 655–67.

[9] Christer Ericson. 2004. *Real-Time Collision Detection.* CRC Press, Inc., Boca Raton, FL, USA.

[10] Nariman Farsad and Andrea Goldsmith. 2015. A Novel Molecular Communication System Using Acids, Bases and Hydrogen Ions. *IEEE International workshop on Signal Processing advances in Wireless Communications* abs/1511.08957. arXiv:1511.08957

[11] N. Farsad, D. Pan, and A. Goldsmith. 2017. A Novel Experimental Platform for In-Vessel Multi-Chemical Molecular Communications. *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. https://doi.org/10.1109/GLOCOM.2017.8255058

[12] Nariman Farsad, Huseyin Birkan Yilmaz, Andrew W. Eckford, Chan-Byoung Chae, and Weisi Guo. 2016. A Comprehensive Survey of Recent Advancements in Molecular Communication. *IEEE Communications Surveys & Tutorials* 18 (2016), 1887–1919. Issue 3.

[13] Luca Felicetti, Mauro Femminella, and Gianluca Reali. 2012. A simulation tool for nanoscale biological networks. *Nano Communication Networks* 3, 1 (2012), 2–18. https://doi.org/10.1016/j.nancom.2011.09.002

[14] Luca Felicetti, Mauro Femminella, and Gianluca Reali. 2013. Simulation of Molecular Signaling in Blood Vessels: Software Design and Application to Atherogenesis. *Nano Communication Networks* 4, 3 (2013), 98–119. arXiv:1306.0150

[15] Luca Felicetti, Mauro Femminella, Gianluca Reali, Paolo Gresele, and Marco Malvestiti. 2013. Simulating an in vitro experiment on nanoscale communications by using BiNS2. *Nano Communication Networks* 4, 4 (2013), 172 – 180.

[16] Ertan Gul, Baris Atakan, and Ozgur B. Akan. 2010. NanoNS: A nanoscale network simulator framework for molecular communications. *Nano Communication Networks* 1, 2 (2010), 138–156. https://doi.org/10.1016/j.nancom.2010.08.003

[17] Vahid Jamali, Nariman Farsad, Robert Schober, and Andrea Goldsmith. 2017. Diffusive Molecular Communications with Reactive Signaling. *IEEE International Conference on Communications (ICC)*. https://doi.org/10.1109/icc.2017.7996907

[18] Yubing Jian, Bhuvana Krishnaswamy, Caitlin M. Austin, A. Ozan Bicen, Jorge E. Perdomo, Sagar C. Patel, Ian F. Akyildiz, Craig R. Forest, and Raghupathy Sivakumar. 2016. nanoNS3: Simulating Bacterial Molecular Communication Based Nanonetworks in Network Simulator 3. In *Proceedings of the 3rd ACM International Conference on Nanoscale Computing and Communication (NANOCOM'16)*. ACM, New York, NY, USA, Article 17, 7 pages. https://doi.org/10.1145/2967446.2967464

[19] Javor Kalojanov, Markus Billeter, and Philipp Slusallek. 2011. Two-Level Grids for Ray Tracing on GPUs. In *EG 2011 - Full Papers*, Oliver Deussen Min Chen (Ed.). Eurographics Association, Llandudno, UK, 307–314. https://doi.org/10.1111/j.1467-8659.2011.01862.x

[20] Javor Kalojanov and Philipp Slusallek. 2009. A Parallel Algorithm for Construction of Uniform Grids. In *Proceedings of the Conference on High Performance Graphics 2009 (HPG '09)*. ACM, New York, NY, USA, 23–28. https://doi.org/10.1145/1572769.1572773

[21] Mehmet S. Kuran, Huseyin Birkan Yilmaz, Tuna Tugcu, and Ian F. Akyildiz. 2011. Modulation Techniques for Communication via Diffusion in Nanonetworks. *2011 IEEE International Conference on Communications (ICC)* (2011), 1–5.

[22] Anthony J. C. Ladd, Hu Gang, J. X. Zhu, and D. A. Weitz. 1995. Time-Dependent Collective Diffusion of Colloidal Particles. *Phys. Rev. Lett.* 74 (Jan 1995), 318–321. Issue 2. https://doi.org/10.1103/PhysRevLett.74.318

[23] Ignacio Llatser, Deniz Demiray, Albert Cabellos-Aparicio, D. Turgay Altilar, and Eduard Alarcón. 2014. N3Sim: Simulation framework for diffusion-based molecular communication nanonetworks. *Simulation Modelling Practice and Theory* 42 (2014), 210 – 222. https://doi.org/10.1016/j.simpat.2013.11.004

[24] Jan Wassenberg and Peter Sanders. 2011. Engineering a Multi-core Radix Sort.. In *Euro-Par (2) (Lecture Notes in Computer Science)*, Emmanuel Jeannot, Raymond Namyst, and Jean Roman (Eds.), Vol. 6853. Springer, 160–169. http://dblp.uni-trier.de/db/conf/europar/europar2011-2.html#WassenbergS11